APPENDIX A

```
CREATE procedure gs_calculate_pacing_factor as

-- determine pacing factor to cap out the account in X days (currently 12)

-- determine the number or stats days remaining
select l.listing_id, 1.0000 pacing_factor, datediff(dd, b.transaction_date, dateadd(dd,-
5,li.NextBillDate)) days_in_cycle, b.transaction_cap, b.transaction_cap - (b.previous_revenue +
allowed_revenue) cap_remaining, datediff(dd,min(c.transaction_date), max(c.transaction_date))
days_total, sum((c.transaction_count+c.transaction_no_count)*c.transaction_rate)
actual_revenue_total,
sum((c.transaction_count+c.transaction_no_count)*c.transaction_rate)/(1+datediff(dd,min(c.transact
ion_date), max(c.transaction_date))) avg_daily_revenue,
sum(c.transaction_count+c.transaction_no_count) transaction_count_total,
sum(c.transaction_count+c.transaction_no_count)/(1+datediff(dd,min(c.transaction_date),
max(c.transaction_date))) transaction_count_day into #pf
from gs_listings l (NOLOCK), CityData..LineItem li (NOLOCK),
CityData..LineItemBillingTransaction b (NOLOCK), CityData..LineItemBillingTransactionDetail c
(NOLOCK)
where b.LineItemId = li.LineItemId and b.transaction_date = (select max(transaction_date) from
gs_transaction_dates)
and li.comp_id = l.comp_id
and li.ProductId = 347
and li.LineItemStatusCode = 'ACTIVE'
and b.transaction_cap > 0
and li.NextBillDate is not null
and c.LineItemId = b.LineItemId
and c.transaction_date >= dateadd(dd,-20,(select max(transaction_date) from gs_transaction_dates))
group by l.listing_id,b.transaction_cap, datediff(dd, b.transaction_date, dateadd(dd,-
5,li.NextBillDate)), b.transaction_cap - (b.previous_revenue + allowed_revenue)

create unique index pfjunkindex on #pf (listing_id)

-- if there are more than 12 days then calculate it for 12 days
update #pf set days_in_cycle = 12 where days_in_cycle <1 or days_in_cycle > 12
update #pf set cap_remaining = 0.0 where cap_remaining < 1.0

--  calculate the pacing factor based on the cap remaining
update #pf set pacing_factor = cap_remaining / (days_in_cycle *(avg_daily_revenue+0.01)) from
#pf where cap_remaining / (days_in_cycle *(avg_daily_revenue+0.01))  < 1.0

-- update the pacing factor for ads not in the current pacing set
update gs_ads set transaction_pacing_factor = 1.0, transaction_sync = 'Y'  where
transaction_pacing_factor < 1.0 and ad_id not in (
select a.ad_id from gs_ads a(NOLOCK), gs_online o (NOLOCK), #pf p (NOLOCK) where
o.listing_id = p.listing_id
and o.online_id = a.online_id and a.ad_id = gs_ads.ad_id)
```

```
-- update the pacing factor for the current set of ads
update gs_ads set transaction_pacing_factor = p.pacing_factor, transaction_sync = 'Y'
from #pf p(NOLOCK), gs_online o (NOLOCK) where o.online_id = gs_ads.online_id
and o.listing_id = p.listing_id and (abs(gs_ads.transaction_pacing_factor - p.pacing_factor) > 0.02)
```

```
CREATE procedure gs_calculate_yield as

/**

calculate yield on a weighted 40 day basis and a 5 day basis, blend the rates on a weighting and
determine transaction yield

the days for the yield weighting is arbiturary

**/

declare @date datetime

select @date = (select max(Date) from CustomerStats..AdUniqueUserFact)

-- get stats data
select i.ad_id, sum(i.ImpressionCount) impressions,
sum(case when i.Date >= dateadd(dd,-7,@date) then i.ImpressionCount else 0 end)
fast_impressions, 10000000 clicks, 10000000 fast_clicks, 100.0000 transaction_yield into #ads from
CustomerStats..AdImpressionFact i (NOLOCK)
where i.Date > dateadd(dd,-40,@date) and i.Date >= '6/19/2003'
group by i.ad_id

create unique index junkads on #ads (ad_id)

update #ads set clicks = 0, fast_clicks =0

select u.ad_id, sum(u.UniqueUser) clicks, sum(case when u.Date >= dateadd(dd,-7,@date) then
u.UniqueUser else 0 end) fast_clicks into #clicks from CustomerStats..AdUniqueUserFact u
where u.Date > dateadd(dd,-40,@date) and u.Date >= '6/19/2003'
group by u.ad_id

update #ads set clicks = c.clicks, fast_clicks = c.fast_clicks from #clicks c (NOLOCK) where
c.ad_id = #ads.ad_id

-- calculate the wieghted yield
update #ads set transaction_yield = (((clicks+1)*1.0)/((impressions+40)*1.0)*0.4 +
((fast_clicks+1)*1.0)/((fast_impressions+40)*1.0)*0.60)* 100.0

-- update the ads with the update tield
```

```
update gs_ads set transaction_yield = a.transaction_yield, transaction_sync =  'Y' from #ads a where
a.ad_id = gs_ads.ad_id
and abs(a.transaction_yield - gs_ads.transaction_yield) >= 0.1
and gs_ads.ProductComponentId in  (311,312,313,314)

-- plant new ads in a good value and let the stats bring the ads down or push the ads up higher

update gs_ads set transaction_yield = 2.3, transaction_sync =  'Y'  where transaction_yield = 100.0
and ProductComponentId in  (311,312,313,314)
```

Search Engine
Ad Globs

```
    package Guide::Model::AdGlobs;

    use strict;
    use base qw(
                Control::Controller
                Guide::Control::SuperSearch::Base
                Control::Search::AnyWhere
        );
    use Guide::Model::PageTypes;
    use Guide::Model::SiteTargets;
    use Guide::Model::AdLimits;
    use Guide::Model::FeaturedAd;
    use Control::Search::AnyWhere;
    use Search::Model::Query;

    ## Distance constant -- used for calculations
    use constant DISTANCE_CONSTANT => 1;

    ## Ads to pull from slicware
    use constant AD_LIMIT => 30;

    sub new {
      my $proto = shift;
      my $class = ref($proto) || $proto;
      my %args = @_;
      my $self = bless (\%args, $class);

      $self;
    }

    sub get_data {
      my $self   = shift;
      my $params = shift;

      return undef unless (ref($params));
      my $ad_data;
```

```
my $search_params = {
  cs_search    => 'pfp_ads',
  rpp          => AD_LIMIT(),
};

my $rkw = {}; # Restrictive Keywords

## let's go get some data about what I'm looking at
## Site target ids
my $page_type_id = &Guide::Model::PageTypes::name_to_id($params->{page_type});
my ($site_target_id, $min_dist, $max_dist, $sales_cats) =
&Guide::Model::SiteTargets::name_to_id({
                    page_type_id => $page_type_id,
                    page_type_data => $params->{page_type_data},
                    });

unless ($site_target_id) {
  warn "No site target found! Page type: " . $params->{page_type} .
     " Page type data: " . $params->{page_type_data} . " -- no ads served";
  return undef;
}

$rkw->{'stid_' . $site_target_id} = 1;
$ad_data->{site_target_id} = $site_target_id;
$ad_data->{sales_cats}     = $sales_cats;
my $multiplier = $params->{market_multiplier} || 1;

## Market / Point radius searching
if ($params->{lat} and $params->{long}) {
  $search_params->{lat}  = $params->{lat};
  $search_params->{long} = $params->{long};
  $search_params->{miles} = ($max_dist * $multiplier);
  $ad_data->{metro_mode} = 0;
}
  else {
  $rkw->{'market_' . $params->{market_id}} = 1;
  $ad_data->{metro_mode} = 1;
  $search_params->{sorted} = 'cpcctr-desc';
}

$self->log('Mode: ');
if ($ad_data->{metro_mode} == 1) {
  $self->log('Metro');
} else { $self->log('Coverage Area'); }
unless ($ad_data->{metro_mode}) {
  $self->log('Sales Categories: ' . $sales_cats);
  $self->log('Center point lat: ' . $search_params->{lat} . ' Long: ' . $search_params->{long});
  $self->log('Market Multipler: ' . $multiplier );
  $self->log('Sales Category max dist: ' . $max_dist );
  $self->log('Max search dist: ' . $search_params->{miles});
  $self->log("Page type: " . $params->{page_type} . " ID: " . $page_type_id);
  $self->log("Page type data: " . $params->{page_type_data});
 }
 $self->log("------ Begin Ad Data ---------");

 ## Set up the limits
 $ad_data->{limits}                                                                      =
&Guide::Model::AdLimits::ad_restriction_by_page_type($page_type_id);

 # Let's get to rock'n  -- Set up new searcher
 my $search = new Search::Model::Query();
 my $total_ads;
 foreach my $ad_type_id (keys %{$ad_data->{limits}}) {
  $total_ads += $ad_data->{limits}->{$ad_type_id};
 }

 ## Ahh darn rules.  We have to at least attempt to show tier 1 ads according to
 ## the same parameters we show everyone else.
 if ($ad_data->{limits}->{1}) {
  $self->log("Begin search for tier 1 ads");
  my ($lrkw, $lsearch_params);
  $ad_data->{1} = [ ];

  %{$lrkw} = %{$rkw}; %{$lsearch_params} = %{$search_params};
  $lrkw->{'ad_type_1'} = 1;
  $lsearch_params->{sorted} = 'weight,bus_name';
  $lsearch_params->{rpp} = 4;

  ## Push RKW's into a string
  $lsearch_params->{rkw} = join ('+', keys %{$lrkw});
  my $results = $search->search($lsearch_params);
  foreach my $data_hash (@{$results->{list}->[0]->{item}}) {
   my $featured_ad = new Guide::Model::FeaturedAd ( $data_hash );
   $featured_ad->metro_mode($ad_data->{metro_mode});

   $self->log("Found tier 1 ad id:" . $featured_ad->ad_id);
   ## Note, we're not checking the limits here because we've limited the result set to 4 items
above
   push @{$ad_data->{1}}, $featured_ad;
  }
  unless (scalar @{$results->{list}->[0]->{item}}) { $self->log("No tier 1 ads found"); }
  $ad_data->{limits}->{1}  =  scalar  @{$ad_data->{1}}  if  (scalar  @{$results->{list}->[0]-
>{item}}
                          and scalar @{$ad_data->{1}} > $ad_data->{limits}->{1});
  $self->log("End search for tier 1 ads");
 }

 ## We're on a two ad system now, and if it's for tier 1, we'll fill it in later
```

```
  $rkw->{'ad_type_2'} = 1;

  ## Push RKW's into a string
  $search_params->{rkw} = join ('+', keys %{$rkw});
  my $results = $search->search( $search_params);
  $self->log("Search Params: \n" . Dumper($search_params)); use Data::Dumper;
  $self->log("Begin search for PFP ads (tier 2)");
  $self->log(sprintf('%5s %30s %5s %8s %8s %6s %5s', 'id','name','dist', 'cpcctr','Pacev', 'Pacet',
'rank'));

  my (@all_ads, $dedup);
  ## Retrieve
  foreach my $data_hash (@{$results->{list}->[0]->{item}}) {
   my $featured_ad = new Guide::Model::FeaturedAd ( $data_hash );
   $featured_ad->metro_mode($ad_data->{metro_mode});

   ## Pacing Factor removal
   my $rand = int(rand(1000));
   if ($featured_ad->pacing == 1000 or $featured_ad->pacing == 1 or $featured_ad->pacing >
$rand) {
     ## Perform sorting calc here
     $featured_ad->rank_number($featured_ad->cpcctr       /       ($featured_ad->dist       +
DISTANCE_CONSTANT()))
       unless ($ad_data->{metro_mode});

     ## De-dup code
     my $key = join ('|' , map {$featured_ad->get_data->{urls}->{$_}->{url_text}
                         if exists $featured_ad->get_data->{urls}->{$_}
                         and exists $featured_ad->get_data->{urls}->{$_}->{url_text}
                        } (qw(business_name tagline)));
     if ($dedup->{$key}) {
      if ($dedup->{$key}->[1] < $featured_ad->rank_number) {
       $self->log(sprintf('%5d %30s % 2.2f % 4.3f % 4.3f % 4.3f % 5.0f -- replacing current',
           $featured_ad->ad_id,     substr($featured_ad->get_data->{urls}->{business_name}-
>{url_text}, 0, 30),
           $featured_ad->dist, $featured_ad->cpcctr/1000, $featured_ad->pacing/1000,
           $rand/1000, $featured_ad->rank_number));
       splice @all_ads, $dedup->{$key}->[0], 1;
       $dedup->{$key} = [scalar @all_ads, $featured_ad->rank_number];
       push @all_ads, $featured_ad;
      }
       else {
       $self->log(sprintf('%5d %30s % 2.2f % 4.3f % 4.3f % 4.3f % 5.0f -- duplicate, lower
value',
           $featured_ad->ad_id,     substr($featured_ad->get_data->{urls}->{business_name}-
>{url_text}, 0, 30),
           $featured_ad->dist,    $featured_ad->cpcctr/1000,    $featured_ad->pacing/1000,
$rand/1000,
           $featured_ad->rank_number));
      }
     }
      else {
      $dedup->{$key} = [scalar @all_ads, $featured_ad->rank_number];
      $self->log(sprintf('%5d %30s % 2.2f % 4.3f % 4.3f % 4.3f % 5.0f',
            $featured_ad->ad_id,      substr($featured_ad->get_data->{urls}->{business_name}-
>{url_text}, 0, 30),
            $featured_ad->dist,      $featured_ad->cpcctr/1000,      $featured_ad->pacing/1000,
$rand/1000, $featured_ad->rank_number));
      push @all_ads, $featured_ad;
     }

    }
     else {
     $self->log(sprintf('%5d %30s % 2.2f % 3.3f % 4.3f % 4.3f % 5.0f -- dropped',
            $featured_ad->ad_id,      substr($featured_ad->get_data->{urls}->{business_name}-
>{url_text}, 0, 30),
            $featured_ad->dist,      $featured_ad->cpcctr/1000,      $featured_ad->pacing/1000,
$rand/1000, $featured_ad->rank_number));
    }

   }

  ## Sort -- no need in metro, it's all good
  unless ($ad_data->{metro_mode}) {
   @all_ads = sort {$b->rank_number <=> $a->rank_number} @all_ads;
  }

$self->log('------------------ Resorted values -----------------');
  foreach my $featured_ad (@all_ads) {
     $self->log(sprintf('%5d %30s % 2.2f % 4.3f % 6.1f %6.0f',
            $featured_ad->ad_id,      substr($featured_ad->get_data->{urls}->{business_name}-
>{url_text}, 0, 30),
            $featured_ad->dist,      $featured_ad->cpcctr/1000,      $featured_ad->pacing/1000,
$featured_ad->rank_number));
  }

  foreach my $ad_type_id (keys %{$ad_data->{limits}}) {
   ## Create the place where I stuff data
   $ad_data->{$ad_type_id} = [] unless ($ad_data->{$ad_type_id});

   while (scalar @{$ad_data->{$ad_type_id}} < $ad_data->{limits}->{$ad_type_id}
         and scalar @all_ads) {
    push @{$ad_data->{$ad_type_id}}, shift @all_ads;
   }

   ## Ranking, and recalc of urls, final prep work
   for (my $i = 0; $i < scalar @{$ad_data->{$ad_type_id}}; $i++) {
    $ad_data->{$ad_type_id}->[$i]->site_target_id($site_target_id);
```

```
        $ad_data->{$ad_type_id}->[$i]->total_ads(scalar @{$ad_data->{$ad_type_id}});
        $ad_data->{$ad_type_id}->[$i]->rank($i+1);
        $ad_data->{$ad_type_id}->[$i]->ad_type_id($ad_type_id);
        $ad_data->{$ad_type_id}->[$i]->recalc_urls;
        splice @{$ad_data->{$ad_type_id}}, $i, 1, $ad_data->{$ad_type_id}->[$i]->get_data;
       }

      delete $rkw->{'ad_type_' . $ad_type_id};
     }
     $self->log('---- End ad data ----');
     return $ad_data;

    }

    sub log {
     my $self = shift;
     my $log  = shift;

     $self->{_log} .= '[' . localtime (time) . '] ' . $log . "\n" if ($log);
     return $self->{_log};
    }

1;
```

<u>Featured Ad</u>

```
    package Guide::Model::FeaturedAd;

    use strict;

    use Util::DBConn;
    use CSConf qw($GUIDE_DB $CSROOT);
    use Guide::Model::LinkNames;
    use BerkeleyDB;
    use constant MAP_LINK_TYPE_ID   => 'map';
    use constant EMAIL_LINK_TYPE_ID => 'email';
    use constant PROFILE_LINK_TYPE_ID => 'profile';

    BEGIN {
     no strict 'refs';
     my @local_methods = qw(
                 rank
                 site_target_id
                 cluster_id
                 weight
                 dist
                 pacing
                 metro_mode
                 ad_type_id
                 ad_id
```

```
                total_ads
                cpcctr
                pacing
                rank_number
               );
  map { my $method = $_;
      *{$method} = sub {
                my $self = shift;
                my $data = shift;
                $self->{$method} = $data if $data;
                $self->{$method};
                    }
       unless defined &{$method};
    } @local_methods;
}

sub new {
  my $class      = shift;
  my $params     = shift;
  return undef unless (ref($params) and $params->{ad_id});
  bless $params, $class;
  $params->process_slicware;
  return $params;
}

sub process_slicware {
  my $self = shift;

  my $obj = {};
  my  @text_data_format  =  qw(entity_id  address_line_1  city  state  zip  phone  image_link
map_is_displayed
               email_address);
  my @multi_valued_fields = qw(ad_bullet_1 ad_bullet_2 ad_bullet_3 website_link reservation);

  my $link_type_ids = {
   cs_offers     => 18,
   respond_rfq   => 19,
   ad_bullet_1   => 25,
   ad_bullet_2   => 26,
   ad_bullet_3   => 27,
   business_name => 28,
   tagline       => 29,
   website_link  => 30,
   reservation   => 31,
  };
  $obj->{ad_id} = $self->{ad_id};

  my @text_data = split ('\|', $self->{textdata});
  map {$obj->{$_} = shift @text_data} @text_data_format;
map {
   my ($url_text, $is_linkable) = (shift @text_data, shift @text_data);
   if ($url_text) {
     $obj->{urls}->{$_} = {
       url_text => $url_text,
       };
     $obj->{urls}->{$_}->{_link_type_id} = $link_type_ids->{$_} if ($is_linkable);
    }
  } @multi_valued_fields;

## Handle 'special' fields -- fields that have been seperated from the text block in slicware
my $slicware_to_urls ={
  ptbtl    => 'tagline',
  bus_name => 'business_name',
};
foreach my $slicware_key (keys %{$slicware_to_urls}) {
  my $reg_name = $slicware_to_urls->{$slicware_key};
  my @data = split ('\|', $self->{$slicware_key});
  my ($url_text, $is_linkable) = (shift @data, shift @data);
  if ($url_text) {
    $obj->{urls}->{$reg_name} = {
      url_text => $url_text,
    };
    $obj->{urls}->{$reg_name}->{_link_type_id}     =     $link_type_ids->{$reg_name}     if
($is_linkable);
  }
}

## Handle special cases
if ($self->{latitude} and $self->{longitude} and $obj->{map_is_displayed} eq 'y') {
  $obj->{urls}->{'map'} = {
          url_text => 'Map',
          _link_type_id => MAP_LINK_TYPE_ID
          };
}

if ($obj->{entity_id} and $obj->{email_address}) {
  $obj->{urls}->{'email'} = {
          url_text => 'Email',
          _link_type_id => EMAIL_LINK_TYPE_ID,
          };
}

$obj->{urls}->{'profile'} = {
          url_text => 'Overview',
          _link_type_id => PROFILE_LINK_TYPE_ID,
          } if ($obj->{entity_id});

## End special cases
```

```
  $self->recalc_urls ($obj);
  $self->{_data_obj} = $obj;

}

sub get_data {
  my $self = shift;
  return $self->{_data_obj} if ($self->{_data_obj});
  return $self->_get_data;
}

sub _get_data {
  my $self = shift;
  my $dbh = Util::DBConn->new ($GUIDE_DB);
warn "Retrieving data from database";

  my $main_sql = "SELECT address_line_1, entity_id, city, state, zip, phone,
                        decode(image_ak_url,NULL, image_external_url, image_ak_url) image_link,
                        email_address
                FROM featured_ads
                WHERE featured_ad_id = ?
                AND ( (image_internal_url IS NOT NULL and image_external_url IS NOT NULL)
                    OR image_internal_url IS NULL)";
  my $links_sql = "SELECT link_type_id, url_text, url
                 FROM featured_ad_urls
                 WHERE featured_ad_id = ?
                 AND   is_displayed = 'y'";

  my $main_sth  = $dbh->prepare($main_sql);
  my $links_sth = $dbh->prepare($links_sql);

  $main_sth->execute($self->{ad_id});
  my $data_hash = $main_sth->fetchrow_hashref('NAME_lc') || {};
  my $obj = {
    %{$self},
    %{$data_hash}
    };

  $links_sth->execute($self->{ad_id});
  while (my $data_row = $links_sth->fetchrow_hashref('NAME_lc')) {
    # We need to rewrite the url at this point to redirect though the redirector
    if ($data_row->{url}) {
      $data_row->{_link_type_id} = $data_row->{link_type_id};
    }
    $obj->{urls}->{&Guide::Model::LinkNames::id_to_name($data_row->{link_type_id})}    =
$data_row;
    delete $data_row->{link_type_id};
  }

  ## Handle special cases
  if ($obj->{latitude} and $obj->{longitude}) {
    $obj->{urls}->{'map'} = {
            url_text => 'Map',
            _link_type_id => MAP_LINK_TYPE_ID
            };
  }

  if ($obj->{entity_id} and $obj->{email_address}) {
    $obj->{urls}->{'email'} = {
            url_text => 'Email',
            _link_type_id => EMAIL_LINK_TYPE_ID,
            };
  }

  $obj->{urls}->{'profile'} = {
            url_text => 'Overview',
            _link_type_id => PROFILE_LINK_TYPE_ID,
            };

  ## End special cases

  $obj->{ad_id} = $self->{ad_id};
  $self->{_data_obj} = $obj;
  $self->recalc_urls($obj);
  return $obj;
}

sub get_url {
  my $self = shift;

  return undef unless ($self->{ad_id} and $self->{link_type_id});
  my $dbh = Util::DBConn->new ($GUIDE_DB);

  ## Handle special cases
  if (  $self->{link_type_id} eq EMAIL_LINK_TYPE_ID
     or $self->{link_type_id} eq MAP_LINK_TYPE_ID
     or $self->{link_type_id} eq PROFILE_LINK_TYPE_ID) {
    my $link_sql = "SELECT latitude, longitude, entity_id
              FROM featured_ads
              WHERE featured_ad_id = ?";
    my $link_sth = $dbh->prepare_cached($link_sql);
    $link_sth->execute( $self->{ad_id} );
    my $data_row = $link_sth->fetchrow_hashref('NAME_lc');
    $link_sth->finish;

    if ($self->{link_type_id} eq EMAIL_LINK_TYPE_ID) {
```

```
    return '/email?id=' . $data_row->{entity_id};
   }
   if ($self->{link_type_id} eq MAP_LINK_TYPE_ID) {
    return "/map?mode=geo&map_lat=" . $data_row->{latitude} . '&map_lon='. $data_row-
>{longitude};
   }
   if ($self->{link_type_id} eq PROFILE_LINK_TYPE_ID) {
    return '/profile/' . $data_row->{entity_id};
   }

  }

  my $link_sql = "SELECT url
           FROM featured_ad_urls
           WHERE featured_ad_id = ?
           AND   link_type_id = ?";

  my $link_sth = $dbh->prepare($link_sql);
  $link_sth->execute($self->{ad_id}, $self->{link_type_id});
  my ($url) = $link_sth->fetchrow;
  $link_sth->finish;
  return $url;
}

sub recalc_urls {
  my $self = shift;
  my $data_obj = shift || $self->get_data;
  foreach my $key (keys %{$data_obj->{urls}}) {
   my $data_row = $data_obj->{urls}->{$key};
   delete $data_row->{url} and next unless ($data_row->{_link_type_id});
    $data_row->{url} = '/redirect/?aid=' . $self->{ad_id} . '&ltid=' . $data_row-
>{_link_type_id};
    $data_row->{url} .= '&eid=' . $data_obj->{entity_id} if $data_obj->{entity_id};
    $data_row->{url} .= '&stid=' . $self->site_target_id if $self->site_target_id;
    $data_row->{url} .= '&clid=' . $self->cluster_id if $self->cluster_id;
    $data_row->{url} .= '&rank=' . $self->rank if ($self->rank);
    $data_row->{url} .= '&dist=' . $self->dist if ($self->dist and not $self->metro_mode);
    $data_row->{url} .= '&atid=' . $self->ad_type_id if ($self->ad_type_id);
    $data_row->{url} .= '&total_ads=' . $self->total_ads if ($self->total_ads);
  }
  return $data_obj;
}


=pod
# perhaps useful later
sub get_url {
  my $self = shift;
  return undef unless ($self->{ad_id} and $self->{link_type_id});
  my $bdb = BerkeleyDB::Hash->new(
                              '-Filename' => $bdb_filename,
                              '-Flags'    => 'DB_RDONLY'
                             )
    or warn "FAILURE: cannot open $bdb_filename: $! $BerkeleyDB::Error" and return undef;

  my $url;
  $bdb->db_get($self->{ad_id} . ',' . $self->{link_type_id}, $url);
  return $url;
}

=cut

1;
```